Proposta di progetto

Stridescape

Gruppo

Nome del gruppo:	Gruppo Roero
Componenti:	Riccardo Maino Matteo Bussolino

Date

Data di sottomissione della proposta di progetto	17/06/2024
Data di accettazione della proposta di progetto	25/06/2024

Descrizione breve

L'applicazione permette di tracciare le proprie camminate, consentendo di monitorare i passi, le calorie, i chilometri e il tempo totale dell'attività. Il percorso effettuato è visibile su mappa. E' possibile consultare dei grafici con le statistiche settimanali, mensili ed annuali delle principali statistiche monitorate. E' presente anche una cronologia delle camminate con tutte le relative statistiche e permette di visualizzare a posteriori i percorsi effettuati su una mappa.

Definizioni

Termine	Definizione
MVVM	Model-View-ViewModel, pattern architetturale software che separe le responsabilità tra le tre componenti principali.
CA	Clean Architecture, architettura software per la progettazione di applicazioni modulari, mantenibili e testabili
UC	Use Case, classi che eseguono singole operazioni di business logic
DI	Dependency Injection, pattern di progettazione che consente di ottenere un decoupling delle componenti di un'applicazione fornendo loro le dipendenze di cui hanno bisogno in modo dinamico
DAO	Data Access Object, fornisce le funzionalità necessarie per interagire con il database
FLP	Fused Location Provider API di Google Play Services che semplifica l'accesso ai dati sulla posizione di un dispositivo Android
BR	Broadcast Receiver, è un componente Android progettato per ricevere e rispondere ad eventi di sistema
FS	Foreground Service, è servizio Android che esegue un'operazione visibile all'utente e che può continuare a funzionare anche quando l'utente non interagisce direttamente con l'app.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

SOMMARIO

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

1 CONTESTO DEL PROGETTO

1.1 Situazione attuale

Attualmente esistono diverse app per il fitness e il tracciamento delle attività, ma spesso mancano di alcune funzionalità chiave o risultano complesse da utilizzare. Alcune carenze comuni sono:

- Tracciamento incompleto: non tutte le app monitorano tutti i dati rilevanti come passi, calorie, distanza, tempo, velocità e percorso.
- Visualizzazione limitata: la visualizzazione dei dati su grafici e mappe potrebbe essere non intuitiva o carente di dettagli.
- Design non ottimale: interfaccia utente poco user-friendly e che non rispetta i principi del material design.
- Abbonamenti: per sfruttare tutte le funzionalità e rimuovere eventuali pubblicità viene richiesto di pagare un abbonamento

Il problema è quindi la mancanza di una soluzione completa e intuitiva per monitorare e motivare le proprie camminate senza che sia richiesto la sottoscrizione di un abbonamento per sbloccare tutte le funzionalità o che l'interfaccia presenti numerosi banner pubblicitari.

1.2 Benefici e creazione di valore

In un mondo sempre più attento alla salute e al benessere, l'attività fisica assume un ruolo fondamentale. Camminare, in particolare, è un'attività semplice, accessibile a tutti e con notevoli benefici per la salute. Tuttavia, molte persone non sono motivate a camminare regolarmente o non hanno gli strumenti per monitorare i propri progressi.

La nostra idea è di creare un'app user-friendly, ben curata e dotata di tutte le funzionalità essenziali. L'obiettivo è incentivare le persone a condurre uno stile di vita più attivo, senza richiedere un abbonamento. L'app sarà priva di pubblicità, garantendo un prodotto di qualità per chi desidera monitorare le proprie passeggiate quotidiane e monitorare i propri progressi in modo efficace.

L'app è rivolta ad un pubblico ampio, includendo persone di tutte le età e livelli di forma fisica, che desiderano migliorare la propria salute e benessere attraverso le camminate

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

1.3 Obiettivi del progetto

Jetpack Compose	Utilizzato per lo sviluppo dell'interfaccia utente dell'applicazione
Theme	Utilizzato per gestire la personalizzazione del tema dell'applicazione, e perciò utilizzato per impostare lo stile della tipografia e la color palette della light e dark mode
Navigation	Utilizzato per gestire la navigazione tra le varie schermate dell'applicazione, dove per ognuna è associata una specifica route mantenuta e salvata in una classe specifica. La navigation dell'applicazione si suddivide in due principali blocchi di navigazione: onboarding e main
MVVM	Utilizzato come pattern architetturale dell'applicazione in modo da avere una separazione dei tre aspetti fondamentali che compongono un'applicazione (Model, View, ViewModel): il **Model** che si occupa della persistenza dei dati, nel nostro caso Room e Preferences DataStore, la **View** che corrisponde all'interfaccia utente, ed il **ViewModel** che funge da intermediario tra view e model. Questo ci ha permesso di avere una separazione dei concerns ottimale e garantire maggiore riutilizzabilità del codice.
Clean Architecture	Utilizzata come architettura software per la progettazione dell'applicazione. Permette di ottenere maggior "separation of concerns" e dipendenze unidirezionali: le varie componenti dipendono solo da astrazioni e non implementazioni concrete. Ogni feature dell'applicazione ha un package specifico, dove la sua struttura interna è suddivisa in tre layer: data, domain e presentation
Event	Utilizzati per definire gli eventi possibili per ognuna delle varie feature della applicazione. Questi eventi saranno generabili dall'utente tramite l'interazione con l'interfaccia e saranno intercettati dal ViewModel
Dagger Hilt	Utilizzato per effettuare la dependency injection delle varie dipendenze nelle varie classi. L'utilizzo della DI permette di ottenere maggior decoupling, maggiore modularità e maggior riutilizzabilità
Room	Utilizzato per salvare in un database locale SQLite tutti i dati relativi statistici delle camminate e i percorsi di quest'ultime. Inoltre verranno memorizzati nel database gli eventuali aggiornamenti nel tempo dell'obiettivo di passi giornaliero
Repository	Utilizzato per interagire con i vari DAO
Preferences DataStore	Utilizzato per salvare velocemente e localmente alcune informazioni utili per l'applicazione, come il fatto di aver già

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

	mostrato l'on boarding o i principali dati dell'utente: nome, altezza e peso
Compose State	Utilizzato per gestire in modo efficiente e reattivo lo stato dell'interfaccia utente, ottenendo di fatto un'interfaccia dinamica e responsive rispetto ai dati da mostrare.
State Flow	Utilizzato per ottenere un tipo di flow reattivo, che significa che emette valori e aggiornamenti ai suoi subscribers in modo asincrono. Offre quindi un modo efficiente e reattivo per gestire lo stato mutabile. I vari StateFlow definiti nei VM verranno collezionati come Compose State nell'interfaccia. L'utilizzo degli StateFlow ci permette di effettuare aggiornamenti thread-safe.
Flow	Utilizzato per ottenere un tipo di flusso reattivo che rappresenta una sequenza asincrona di valori. Permette di gestire in modo efficiente e strutturato dati che cambiano nel tempo. In particolare nell'applicazione sono utili per ottenere il caricamento dei dati dal database, continuando ad ottenere sequenze di valori aggiornati nel caso in cui avvengono delle modifiche dei dati del database
Coroutines	Utilizzate per eseguire le operazioni in modo asincrono, migliorando drasticamente le performance dell'applicazione. Permettono di avere una fruizione ottimale e senza rallentamenti dell'applicazione, evitando di bloccare il main thread nelle computazioni lunghe
Material3	Utilizzato come principio per sviluppare le varie caratteristiche dell'interfaccia utente, come abbinamento di colori e caratteristiche dei vari components
Notification Manager	Utilizzato per mostrare all'utente una notifica che fornisce informazioni su passi e tempo della camminata in corso. La notifica mostrata è necessaria per il FS per evitare l'uccisione del processo da parte di Android dopo un certo periodo di tempo quando l'applicazione non è più in primo piano.
Foreground Service	Utilizzato per implementare il Tracking Service, che si occuperà di mantenere e gestire il servizio di tracking della posizione dell'utente, del conteggio del tempo e dei vari listener sui sensori utilizzati. L'utilizzo di un Foreground Service è necessario per far rimanere attiva la computazione anche nel momento in cui l'applicazione non si trova in primo piano
Broadcast Receiver	Utilizzato per rispondere all'evento del sistema relativo all'abilitazione o disabilitazione della localizzazione, per andare a compiere certe azioni durante la fase di tracciamento
Step Counter Sensor	Utilizzato per contare i passi dell'utente mentre è in corso una camminata.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Accelerometer Sensor	Sensore alternativo utilizzato nel caso in cui non sia disponibile lo step counter per contare i passi dell'utente durante una camminata.
Fused Location Provider	Utilizzato per ottenere la posizione precisa dell'utente attraverso delle API ottimizzate. Questo sistema permette di combinare il GPS con la rete per ottenere la localizzazione più precisa possibile dell'utente.
Maps SDK	Utilizzata per mostrare la mappa di Google Maps nell'applicazione, mostrando all'utente la sua posizione attuale ed il percorso che sta seguendo, aggiornato in tempo reale. Viene utilizzata anche per mostrare i percorsi passati che l'utente ha effettuato

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

2 PROFILO DEL PROGETTO

2.1 Ambito del progetto

L'applicazione sviluppata ha come obiettivo principale quello di fornire agli utenti un sistema completo per tracciare e monitorare le proprie camminate. Attraverso questa applicazione, gli utenti potranno non solo tenere traccia dei passi effettuati, delle calorie bruciate, dei chilometri percorsi e del tempo totale dell'attività, ma anche visualizzare il percorso effettuato su una mappa interattiva. Inoltre, l'applicazione offre la possibilità di consultare dettagliate statistiche settimanali, mensili ed annuali, permettendo agli utenti di monitorare i loro progressi nel tempo.

L'applicazione utilizza i servizi e le API di Google per fornire una serie di funzionalità avanzate e migliorare l'esperienza dell'utente:

- **Maps SDK:** permette di integrare mappe interattive all'interno dell'applicazione. Gli utenti possono visualizzare il loro percorso sulla mappa, ottenere una visione dettagliata del tragitto effettuato e ricevere informazioni geografiche precise.
- **API di Google:** Per le funzionalità di localizzazione, l'app si avvale del FLP di Google, che fornisce dati di localizzazione accurati e a basso consumo energetico, combinando più fonti di dati come GPS, Wi-Fi e rete cellulare.

Cosa sarà sviluppato:

- **Tracciamento della posizione in tempo reale**: monitoraggio costante della posizione dell'utente durante le passeggiate utilizzando il FLP. Questo permette di ottenere un rilevamento accurato del percorso effettuato e di fornire aggiornamenti in tempo reale sulla mappa.
- **Monitoraggio della durata dell'attività**: registrazione del tempo totale trascorso durante ogni passeggiata. Questa funzionalità consente agli utenti di sapere esattamente quanto tempo hanno dedicato alla loro attività fisica.
- **Calcolo della distanza percorsa**: misurazione con precisione della distanza totale coperta durante ogni passeggiata. Utilizzando i dati GPS e altre fonti di localizzazione, l'applicazione fornisce una stima accurata dei chilometri percorsi.
- **Visualizzazione del percorso sulla mappa interattiva**: visualizzazione del tragitto seguito dall'utente su una mappa interattiva integrata nell'app. Questa funzionalità utilizza il Maps SDK per offrire una rappresentazione visiva chiara e dettagliata del percorso.
- **Monitoraggio e visualizzazione di statistiche dell'attività**: riepilogo completo delle principali metriche di ogni passeggiata, tra cui distanza percorsa, tempo impiegato, calorie bruciate, numero di passi effettuati e velocità media. Le statistiche saranno presentate in modo chiaro e comprensibile, anche attraverso grafici che facilitano la visualizzazione dei progressi nel tempo.
- **Memorizzazione locale dei dati dell'attività**: salvataggio di tutte le informazioni relative alle passeggiate direttamente sul dispositivo dell'utente. Questo assicura che i dati siano sempre accessibili anche senza una connessione internet e permette agli utenti di rivedere storicamente le loro attività.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- **Interfaccia utente intuitiva**: progettazione di un'interfaccia utente che sia non solo facile da usare, ma anche visivamente piacevole. L'uso di Jetpack Compose garantisce una UI moderna e reattiva, migliorando l'esperienza dell'utente.
- **Notifica informativa sull'attività**: notifica durante le passeggiate per fornire aggiornamenti in tempo reale su metriche come tempo trascorso e passi effettuati. La notifica garantisce che l'utente sia sempre informato sullo stato della sua attività anche quando l'app è in background.

Cosa resterà da sviluppare:

- **Funzionalità avanzate:** in futuro, si potrà implementare il supporto per attività multiple (es. corsa, ciclismo), funzionalità social per condividere i percorsi con amici e community, integrazione con servizi Cloud, integrazioni con altri dispositivi wearable e notifiche contestuali con informazioni utili sui punti di interesse vicini.
- **Analisi avanzata dei dati:** potrebbero essere sviluppate funzionalità di analisi più avanzate, come l'identificazione automatica dei percorsi preferiti e suggerimenti per nuovi percorsi.
- **Personalizzazione:** Maggiori opzioni di personalizzazione per le notifiche e le visualizzazioni delle statistiche.

2.2 Profilo della soluzione da realizzare

L'applicazione è sviluppata seguendo il principio della Single Activity Architecture, una metodologia attualmente raccomandata per lo sviluppo di applicazioni Android. Questo approccio prevede che l'intera applicazione sia gestita tramite una singola attività (Activity), la cui interfaccia, ovvero le varie schermate presentate all'utente, siano cambiate dinamicamente facendo utilizzo della libreria di Navigation di Compose. Così come la maggior parte delle applicazioni di questo tipo, l'applicazione verrà configurata per funzionare solo in portrait mode.

L'applicazione sarà suddivisa nelle seguenti schermate:

- **Onboarding**: una fase iniziale che mostrerà all'utente le principali funzionalità che l'applicazione fornisce e permetterà di impostare le informazioni dell'utente (nome, peso, altezza e obiettivo di passi giornaliero)
- **Home**: la pagina principale dell'applicazione. Mostrerà un riepilogo delle statistiche (passi effettuati, chilometri percorsi, tempo trascorso e calorie bruciate) giornaliere. Inoltre verrà visualizzato graficamente un riepilogo dei passi effettuati nell'ultima settimana
- **History**: per visualizzare la cronologia delle passeggiate fatte, mostrando all'utente un riepilogo con le varie statistiche di ogni camminata e la relativa mappa per ogni percorso
- **Tracking**: la pagina visualizzata nel momento in cui si avvia una camminata, che mostrerà la mappa ed una scheda live delle statistiche monitorate. Sarà visibile la posizione attuale sulla mappa ed il tracciato del percorso compiuto fino a quel momento
- **Statistics**: pagina che permette di mostrare i dati su un grafico, in modo da avere una visualizzazione più curata dei dati monitorati. Avremo modo di visualizzare il grafico su intervalli di tempo diversi: settimanale, mensile ed annuale
- **Profile**: la pagina che mostra, e permette di modificare, le informazioni dell'utente.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

3 Requirements, Vincoli e Assunti

3.1 Requirements

1. Requisiti Funzionali:

- **Tracciamento della posizione:** tracciare la posizione dell'utente durante le passeggiate in tempo reale.
- **Tracciamento del tempo:** misurare la durata delle passeggiate.
- **Calcolo della distanza:** determinare la distanza percorsa durante le passeggiate.
- **Visualizzazione della mappa:** mostrare il percorso effettuato su una mappa interattiva.
- **Visualizzazione delle statistiche:** fornire statistiche dettagliate sulle passeggiate, come distanza percorsa, tempo impiegato, calorie bruciate, passi effettuati e velocità media. Queste statistiche dovranno essere mostrate anche su dei grafici per facilitare la visualizzazione.
- **Salvataggio dei dati:** conservare localmente i dati delle passeggiate effettuate.
- **Interfaccia utente:** offrire un'interfaccia utente intuitiva, facile da usare e ben curata.
- **Notifiche:** mostrare all'utente una notifica con alcune informazioni sulla camminata, come tempo e passi.
- **Gestione dei permessi:** deve avvenire una corretta gestione dei permessi che devono essere richiesti all'utente

2. Requisiti Non Funzionali:

- **Prestazioni:** l'app deve essere efficiente e fluida, con tempi di risposta rapidi e bassi consumi di batteria.
- **Affidabilità:** l'app deve essere stabile e affidabile, con crash o malfunzionamenti minimali.
- **Sicurezza:** l'app deve proteggere i dati personali dell'utente in modo sicuro, rispettando le normative sulla privacy qualora venisse utilizzato il cloud.
- **Usabilità:** l'app deve essere facile da usare e intuitiva anche per utenti non esperti di tecnologia, e deve essere conforme alle linee guide del Material Design.
- **Accessibilità:** l'app deve essere accessibile a persone con disabilità.
- **Multilingua:** l'app deve essere disponibile in più lingue per un pubblico globale.
- **Supporto Dark Mode:** l'app deve supportare correttamente la dark mode utilizzando colori coerenti
- **Interfaccia Offline:** l'app deve funzionare in parte anche in modalità offline, per permettere l'utilizzo anche in assenza di connessione internet.

3. Considerazioni Aggiuntive:

- L'app deve essere progettata usando la clean architecture in modo modulare, per facilitare lo sviluppo futuro e la manutenzione.
- L'app dovrebbe seguire le linee guida di design di Android per garantire un'esperienza utente coerente e intuitiva.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- L'app dovrebbe essere testata accuratamente per garantire la sua funzionalità, affidabilità e sicurezza.

3.2 Vincoli temporali

Data	Descrizione
01/05/2024	Inizio progetto
30/06/2024	Prima release funzionante
01/07/2024	Fine progetto: sessione di esame di luglio

3.3 Vincoli tecnologici

I vincoli tecnologici sono i seguenti:

- **Piattaforma di sviluppo**: L'app sarà sviluppata utilizzando Android Studio, l'ambiente di sviluppo integrato (IDE) ufficiale per Android.
- **Linguaggio di programmazione**: L'app sarà scritta in Kotlin, il linguaggio di programmazione consigliato per lo sviluppo Android moderno.
- **Compatibilità**: La versione minima di SDK è la 28 (Android 9.0 Pie) per garantire un equilibrio tra modernità e compatibilità, coprendo una vasta gamma di dispositivi ancora in uso.
- **Testing**: L'app verrà testata all'interno di differenti emulatori Android, e sui devices fisici da noi posseduti in modo tale da avere un'idea più veritiera delle perfomance.

3.4 Permessi

android.permission.INTERNET	Questo permesso è necessario per scaricare la mappa.
android.permission.ACCESS_FINE_LOCATION **android.permission.ACCESS_COARSE_LOCATION** **android.permission.ACCESS_BACKGROUND_LOCATION**	Entrambi questi permessi sono necessari per tracciare la posizione in tempo reale durante le passeggiate e per tracciare la posizione in background per applicazioni con target API level uguale o superiore ad Android 10 (API Level 29)
android.permission.ACTIVITY_RECOGNITION	Questo permesso è necessario per poter accedere al sensore di rilevamento dei passi.
android.permission.FOREGROUND_SERVICE **android.permission.FOREGROUND_SERVICE_LOCATION** **android.permission.FOREGROUND_SERVICE_HEALTH**	Questo permesso è necessario per le applicazioni che utilizzano un FS ed hanno come livello di API target superiore o uguale Android 9 (API level 28). E' anche necessario specificare il tipo di FS, mediante i due permessi aggiuntivi.
android.permission.POST_NOTIFICATIONS	Questo permesso è necessario da Android 13 (API level 33) o superiore per permettere

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

	l'invio di notifiche, ad esempio quelle fornite dal FS per il tracciamento.

3.5 Eventuali assunti

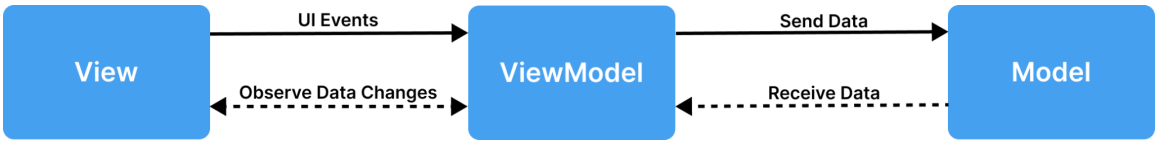
- Assumiamo che per ottenere una corretta visualizzazione della mappa l'utente disponga di una connessione ad internet

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

4 DESIGN

4.1 Architettura MVVM

L'architettura MVVM (Model-View-ViewModel) è stata adottata per mantenere una chiara "separation of concerns", migliore manutenibilità e facilitazione per testing del codice.



Model:

- **Descrizione**: Contiene la logica di business e i dati dell'applicazione.
- **Componenti**:
  - **Use Cases**: Classi che incapsulano la logica di business complessa o semplice che può essere riutilizzata da più ViewModel. Questi permettono di evitare la duplicazione del codice, permettono di ottenere una migliore suddivisione delle responsabilità, migliorano la leggibilità e la testabilità
  - **Entity**: Classi che rappresentano le tabelle del database, nel nostro caso abbiamo utilizzato diverse entità per quanto riguarda le singole camminate, i target giornalieri ed i percorsi.
  - **Repository**: Classi che gestiscono le operazioni di accesso ai dati, fornendo un'interfaccia coerente per ottenere e memorizzare dati relativi a target e camminate
  - **Room Database**: Utilizzato per l'archiviazione locale dei dati, sfruttando la libreria Room di Android Jetpack per una gestione efficiente del database SQLite, in questo modo possiamo memorizzare tutte le camminate con il relativo percorso e le varie metriche tracciate, così come i target giornalieri.
  - **Preferences DataStore:** Utilizzato per salvare i dati dell'utente riguardanti il peso, altezza, e nome.

View:

- **Descrizione**: Comprende tutte le varie screen dell'applicazione, nel nostro caso abbiamo 5 screen principali, una per ogni feature: **OnBoarding**, **Home**, **Statistics, History**, **Profile** e **Tracking**

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- **Componenti**: Ogni screen è composta da diversi componenti, (e.g pulsanti, tiles, cards) che vanno a specificare le varie sezioni del layout. La suddivisione in componenti permette di rendere il codice più leggibile e riutilizzabile.

ViewModel:

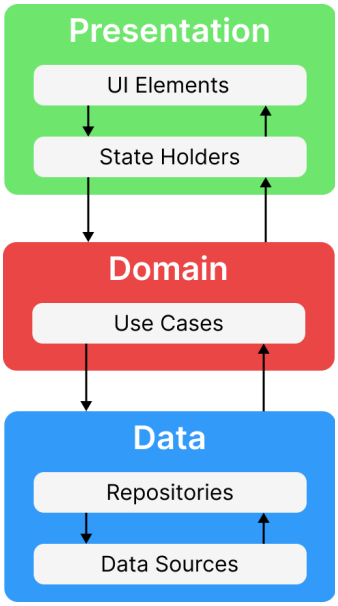
- **Descrizione**: Intermediario tra il Model e la View, gestisce la logica di presentazione e mantiene lo stato dell'interfaccia utente.
- **Componenti**:
  - **StateFlow** e **Compose State**: Utilizzati per aggiornare la UI in modo reattivo quando i dati cambiano.
  - **ViewModel**: Classi che svolgono il ruolo di state holders e che hanno la responsabilità di intercettare gli eventi della UI.

4.2 Clean Architecture

L'applicazione è stata strutturata seguendo i principi della Clean Architecture, per garantire modularità, testabilità, manutenibilità e riutilizzabilità del codice. Abbiamo diversi packages, uno per ogni feature (dove una feature rappresenta per l'appunto una funzionalità offerta) per facilitare la suddivisione del codice. Per ogni feature sono definiti i seguenti layer:

- **Data:** Gestisce l'accesso ai dati, utilizzando repository per incapsulare la logica di recupero e salvataggio dei dati da diverse sorgenti (database locale, preferences datastore).
- **Domain:** Contiene la logica di business principale dell'applicazione, con entità di dominio che rappresentano i concetti chiave (passeggiate, statistiche, etc.) e use case che gestiscono le operazioni specifiche (avviare la passeggiata, terminare la passeggiata, calcolare le statistiche, etc.). Inoltre definisce le interfacce di manager e repositories successivamente implementate nel data layer, così come le classi utilizzate per interagire con i sensori.
- **Presentation:** Comprende la UI e la logica di presentazione. Fa utilizzo dei composable di Jetpack Compose per la definizione dell'interfaccia utente e ViewModel per la gestione dello stato, l'intercettazione degli eventi di interfaccia, e la comunicazione con il livello Domain.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603



4.3 Uso di Jetpack

Il progetto sfrutta diverse librerie e componenti di Android Jetpack per ottimizzare lo sviluppo e assicurare un'applicazione robusta e reattiva.

- **Compose:**
  - **Utilizzo:** Costruzione di interfaccia utente dell'applicazione. Permette di creare interfacce dichiarative e reattive, facilitando lo sviluppo e la manutenzione dell'UI.
  - **Esempio:** Tutte le componenti presenti nell'applicazione sono costruite utilizzando Compose e le composable functions.
- **StateFlow e Compose State**
  - **Utilizzo**: Osservazione dei dati nel ViewModel e aggiornamento automatico della UI quando i dati cambiano. Questo permette di mantenere l'UI in sincronia con i dati.
  - **Esempio**: Quando una nuova camminata viene salvata nel database, lo StateFlow ottenuto dal Repository nell'HomeViewModel viene convertito in un Compose State, che permette di aggiornare la visualizzazione con i nuovi dati nella HomeScreen.
- **Room**
  - **Utilizzo**: Fornisce un'astrazione sul database SQLite, permettendo una gestione più semplice e sicura dell'accesso ai dati. Le entità sono mappate a tabelle nel database, e i DAO (Data Access Object) definiscono le operazioni di accesso ai dati. Essi sono delle interfacce che definiscono i metodi, con le relative query, che definiscono tali operazioni. L'implementazione effettiva di questi metodi sarà gestita automaticamente da Room.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

    - **Esempio**: La classe WalkDao contiene metodi per inserire, aggiornare, eliminare e recuperare dati sulle camminate.
- **ViewModel**
    - **Utilizzo**: Mantiene e gestisce i dati relativi all'interfaccia utente, svolgendo di fatto il ruolo di state holders. In questo modo gli stati mantenuti vengono preservati anche con le "configuration changes", come le rotazioni dello schermo.
    - **Esempio**: L'HistoryViewModel carica e conserva i dati delle camminate passate, permettendo alla HistoryScreen di visualizzare la cronologia. Eventuali cambi di configurazione non necessitano il ricaricamento dei dati dal database.
- **Navigation Component**
    - **Utilizzo**: Gestisce la navigazione all'interno dell'applicazione, facilitando il passaggio tra diverse schermate in modo dichiarativo e permette di passare alle varie screen tutti i parametri necessari dai ViewModel, questo aiuta a mantenere il ViewModel separato dalla View, migliorando di fatto la riusabilità.
    - **Esempio**: La HomeScreen utilizza il Navigation Component per navigare verso TrackingScreen, HistoryScreen e StatisticsScreen.
- **Dagger Hilt**
    - **Utilizzo:** Gestisce la dependency injection, facilitando l'iniezione delle dipendenze nelle classi del progetto, come ViewModel e Repository
    - **Esempio:** Gli Use Cases sono iniettati nei vari ViewModel, questo riduce la complessità del codice e migliorando la modularità e testabilità dell'applicazione.
- **Coroutines**
    - **Utilizzo**: Le coroutines, anche se non sono ufficialmente parte di Jetpack, sono strettamente integrate con diverse librerie Jetpack e sono state utilizzate per gestire operazioni asincrone in modo semplice e leggibile, evitando il cosiddetto "callback hell" e semplificando il codice asincrono.
    - **Esempio**: Nell'HomeViewModel, le coroutines sono utilizzate per eseguire operazioni di ottenimento dei dati da mostrare sulla HomeScreen in modo asincrono.
- **Flow**
    - **Utilizzo:** I flow, anche se non sono ufficialmente parte di Jetpack, sono strettamente integrati con diverse librerie Jetpack e rappresentano un modo per gestire in modo efficiente e strutturato flussi di dati reattivi, rappresentando quindi una sequenza asincrona di valori. Nell'applicazione sono utili per ottenere il caricamento dei dati dal database, continuando ad ottenere sequenze di valori aggiornati nel caso in cui avvengono delle nuove modifiche dei dati del database.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- **Esempio:** Gli Use Case che ottengono i dati dal database, mediante l'utilizzo di repository, vanno a restituire dei flow, in modo tale da far ottenere ai ViewModels che li utilizzano il valore della sequenza più aggiornato.

4.4 Uso Componenti Android

Per lo sviluppo dell'applicazione sono state utilizzate diverse funzionalità fornite da componenti ed API di Android:

- **Foreground Service**
  - **Utilizzo:** Servizio eseguito in background. L'utente è a conoscenza dell'attività in corso attraverso una notifica persistente.
  - **Esempio:** La funzione di tracciamento dell'utente, calcolo del tempo e dei passi è mantenuta operativa anche in background quando l'app non è aperta grazie al Foreground Service relativo al tracciamento.
- **Notification Manager**
  - **Utilizzo:** Gestisce e permette la visualizzazione delle notifiche nell'area di notifica di Android.
  - **Esempio:** L'applicazione mostra all'utente una notifica con il tempo ed i passi effettuati durante la camminata.
- **Sensors**
  - **Utilizzo:** Accesso ai dati rilevati dai sensori presenti nel dispositivo.
  - **Esempio:** Nell'applicazione otteniamo i dati relativi allo Step Counter e l'Accelerometro per poter rilevare e contare i passi dell'utente.
- **Broadcast Receiver**
  - **Utilizzo:** Ricezione e reazione a messaggi broadcast da altre applicazioni o dal sistema operativo
  - **Esempio:** Nell'applicazione è stato utilizzato per rilevare se il GPS è attivo nel dispositivo e, in caso contrario, richiedere all'utente di attivarlo.
- **Fused Location Provider**
  - **Utilizzo:** API di Google Play Services che fornisce una soluzione unificata per ottenere la posizione dell'utente combinando dati da GPS, Wi-Fi e reti mobili, ottenendo un tracciamento molto più preciso e con un basso consumo energetico.
  - **Esempio:** I dati relativo al tracciamento dell'utente sono ottenuti tramite il Fused Location Provider
- **Maps SDK**
  - **Utilizzo:** Fornisce strumenti per integrare mappe interattive nelle applicazioni Android. Questo SDK consente di personalizzare e manipolare le mappe, aggiungere marker, tracciare percorsi e molto altro, offrendo agli utenti una visualizzazione dettagliata delle informazioni geografiche.
  - **Esempio:** Nell'applicazione è possibile visualizzare il tracciamento del percorso della camminata sia in tempo reale, che a posteriori nella sezione della cronologia.

4.5 Soddisfazione dei requisiti

L'architettura MVVM, l'uso delle capacità di Jetpack e l'utilizzo dei componenti Android permettono di soddisfare pienamente i requisiti del progetto in diversi modi:

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- **Interfaccia utente**: Compose ha permesso la creazione dell'UI, agevolando lo sviluppo e la sua manutenzione. Compose consente di aggiornare automaticamente la UI in risposta ai cambiamenti dei dati che osserva.
- **Tracciamento delle metriche**: L'uso del FLP e dei sensori del dispositivo (come lo Step Counter e l'Accelerometro) permette di tracciare con precisione la posizione, i passi e le altre metriche relative all'attività fisica dell'utente. Le metriche sono raccolte e visualizzate in tempo reale, garantendo dati accurati e aggiornati. Il tutto è computato utilizzando un Foreground Service per permettere la raccolta anche quando l'applicazione viene chiusa in fase di tracciamento.
- **Persistenza dei dati**: Room fornisce un'astrazione sicura ed efficiente per la gestione del database, assicurando che i dati siano persistiti correttamente e accessibili anche offline.
- **Navigazione intuitiva**: Il Navigation Component facilita una navigazione coerente e intuitiva, e facilmente estendibile.
- **Gestione delle dipendenze**: Dagger Hilt semplifica l'iniezione delle dipendenze, riducendo il boilerplate code e migliorando la modularità dell'applicazione. Questo rende più facile testare le singole componenti e sostituirle con mock durante gli unit test.
- **Notifiche**: Il Notification Manager gestisce e permette la visualizzazione della notifica durante il tracciamento delle camminate. L'app mostra una notifica persistente che informa l'utente sul tempo trascorso e i passi effettuati, anche quando l'app è in background. La notifica è necessaria per il Foreground Service, per indicare all'utente che il tracciamento è attivo, garantendo che l'utente sia sempre consapevole.
- **Gestione dei permessi:** I permessi vengono gestiti correttamente utilizzando strutture dati per mantenere traccia dei permessi rifiutati. Sono stati definiti dei dialog che vengono mostrati in caso di rifiuto e che permettono di richiedere nuovamente il permesso o di aprire le impostazioni.
- **Manutenibilità**: La chiara "separation of concerns" rende il codice più facile da mantenere e aggiornare. Ogni componente ha una responsabilità specifica, riducendo la complessità e facilitando il debug.
- **Reattività e performance**: Flow, StateFlow, Compose State e Coroutines assicurano che l'applicazione rimanga reattiva anche durante operazioni di accesso ai dati intensive. Le operazioni asincrone evitano blocchi dell'UI thread, migliorando l'esperienza utente.
- **Testabilità**: L'architettura MVVM, la DI e la CA facilitano il testing, poiché i vari componenti possono essere testati in isolamento rispetto alla UI. Le classi Repository possono essere facilmente testate con mock database. Gli UCs e ViewModels sono facilmente in isolamento dagli altri componenti facendo l'injection di classi "fake".

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5 IMPLEMENTAZIONE

5.1 Struttura dell'applicazione

L'applicazione è stata suddivisa in varie features:

- **OnBoarding**: Feature che implementa la fase iniziale che mostra all'utente le principali funzionalità che l'applicazione fornisce e che permette di impostare le informazioni dell'utente (nome, peso, altezza e obiettivo di passi giornaliero)
- **Home**: Feature principale dell'applicazione. Mostrerà un riepilogo delle statistiche (passi effettuati, chilometri percorsi, tempo trascorso e calorie bruciate) giornaliere. Inoltre verrà visualizzato graficamente un riepilogo dei passi effettuati nell'ultima settimana
- **History**: Feature per visualizzare la cronologia delle passeggiate fatte, mostrando all'utente un riepilogo con le varie statistiche di ogni camminata e la relativa mappa per ogni percorso
- **Tracking**: Feature per la visualizzazione dell'interfaccia di tracciamento nel momento in cui si avvia una camminata. Mostrerà la mappa ed una scheda live delle statistiche monitorate. Sarà visibile la posizione attuale sulla mappa ed il tracciato del percorso compiuto fino a quel momento
- **Statistics**: Feature che si occupa di mostrare i dati su un grafico, in modo da avere una visualizzazione più curata dei dati monitorati. La visualizzazione del grafico è gestita su intervalli di tempo diversi: settimanale, mensile ed annuale
- **Profile**: Feature che si occupa di mostrare la pagina utente, e che permette di modificare le informazioni dell'utente

Per ognuna delle feature è stato definito un package. Inoltre, siccome la struttura dell'applicazione è basata sul principio della Clean Architecture, ogni feature è suddivisa in tre layer corrispondenti a tre packages:

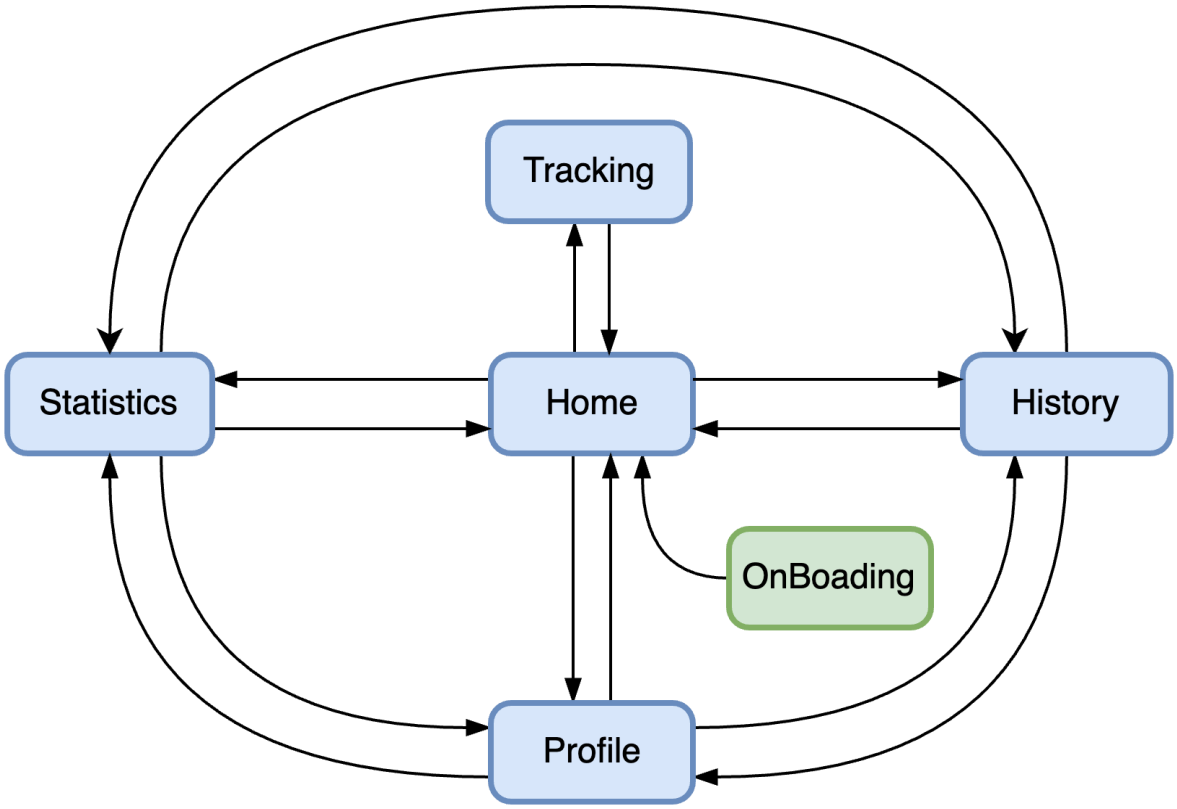
- **Presentation**
- **Domain**
- **Data**

Allo stesso livello della hierarchy dei feature packages è presente un package **core**. Questo è costituito da componenti e/o classi generali oppure componenti e/o classi comuni a più feature. Lo scopo di questo package è quello di agevolare il riuso ed evitare la duplicazione del codice

E' presente anche un package di **Dependency Injection**, che contiene i moduli utilizzati da Dagger Hilt per effettuare la dependency injection. Questo package è fondamentale per garantire una gestione efficiente e modulare delle dipendenze all'interno dell'applicazione. Grazie a Dagger Hilt, possiamo definire come le diverse componenti dell'applicazione, come i repository, gli use case, i database locali e altri elementi, vengono istanziati come **Singleton** e forniti dove necessario.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

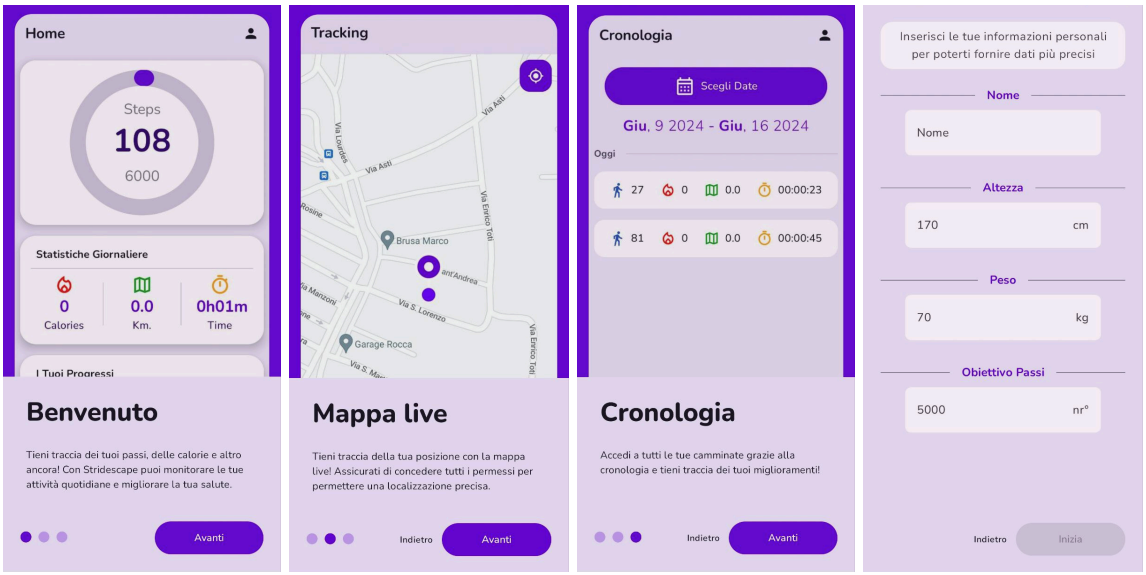
Ogni feature della nostra applicazione include una o più screen. Per gestire in modo efficace la navigazione tra queste screen, abbiamo definito un Navigation Graph, facendo uso della libreria di Navigation di Jetpack. Per migliorare la manutenibilità e facilitare l'estensione del progetto, ad ogni screen è stato associato un oggetto Route. Questo oggetto Route è una sealed class che incapsula la stringa utilizzata come nome della destinazione nel NavHost, garantendo una gestione centralizzata e coerente delle destinazioni di navigazione. Il Navigation Graph definito segue il seguente schema:



Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.2 OnBoarding

Questa feature ha il compito di introdurre l'utente all'applicazione. La schermata associata, la **OnBoardingScreen**, viene mostrata solo al primo avvio dell'applicazione, ed una volta completato l'inserimento dei dati riguardanti l'utente, esso viene reindirizzato alla **HomeScreen**.



L'OnBoarding è principalmente composta da due schermate. La prima schermata mostra all'utente le funzionalità principali dell'applicazione, mentre la seconda richiede l'inserimento dei dati personali riguardanti l'utente, ovvero il nome, il peso, l'altezza e l'obiettivo giornaliero di passi.

Per gestire la visualizzazione delle funzionalità offerte dell'applicazione nell'OnBoardingScreen, viene utilizzato un **Pager**, dove ciascuna schermata è definita tramite una data class **Page**, che contiene le informazioni della pagina da mostrare come il titolo, la descrizione e l'immagine.

La schermata per l'inserimento dei dati personali consiste in una serie di campi di testo , i quali, una volta modificati, lanciano un evento di tipo **OnBoardingEvent** mediante la funzione del **OnBoardingViewModel** che si occupa della gestione degli eventi UI relativi a questa feature. La funzione è chiamata **onEvent()**, ed è l'unica funzione esposta all'UI e perciò è utilizzata per la gestione di una qualsiasi interazione dell'utente. In questo caso, tale funzione richiama le corrispondenti funzioni private del OnBoardingViewModel, che andranno a salvare il nuovo

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

valore nel **Compose State**, che contiene un oggetto di tipo **UiOnBoardingState**. Il suo compito è quello di mantenere i dati e gli eventuali errori riguardanti i vari campi della OnBoardingScreen. Viene anche eseguita una validazione del testo inserito richiamando gli specifici **Use Case**, per controllare la presenza di eventuali errori. La presenza di un errore comporta anch'esso l'aggiornamento del Compose State, in modo tale da scatenare una recomposition per mostrare l'errore all'utente. Tutto ciò è eseguito in una coroutine, in modo da evitare il blocco dell'UI Thread.

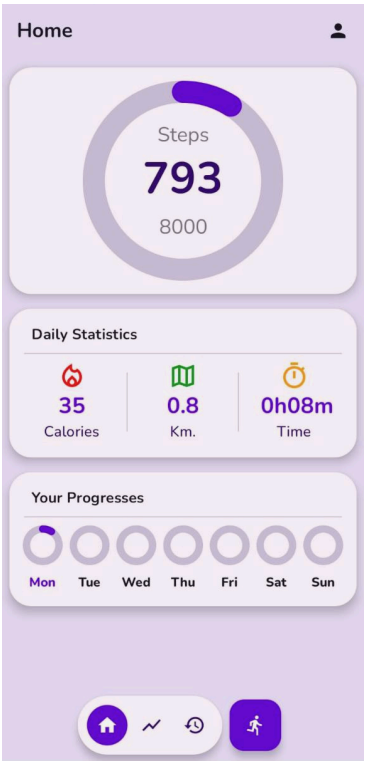
Quando non ci sono errori e l'utente procede premendo il pulsante "Get Started", viene lanciato un nuovo evento, il quale avvierà una serie di coroutines che si occuperanno di salvare le informazioni dell'utente nel **DataStore** e l'obiettivo di passi giornaliero nel **Database**, il tutto eseguendo gli opportuni **Use Cases**.

Il salvataggio dei dati nel DataStore include anche l'informazione relativa al fatto che il processo di onboarding è stato completato. Questa informazione verrà prelevata e consultata ad ogni avvio dell'app, in modo da determinare se reindirizzare direttamente l'utente alla Home feature, saltando il processo di onboarding.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.3 Home

Dopo il primo avvio dell'app, l'utente viene accolto dalla schermata iniziale, la **HomeScreen**. Questa feature, pensata per offrire un'esperienza immediata e informativa, raccoglie e visualizza una serie di dati relativi alle attività dell'utente.



Per popolare la HomeScreen con i dati pertinenti, l'app utilizza una serie di **Use Case**, ovvero componenti software dedicati a specifiche operazioni. Ogni Use Case interagisce con un **Repository** specifico, un elemento che funge da intermediario tra l'app e i dati persistenti (come un database). Per i dati relativi alle camminate verrà utilizzato il **WalkRepository**, mentre per quanto riguarda il recupero delle informazioni degli obiettivi giornalieri verrà utilizzato il **TargetRepository**.

I Repository restituiscono dei dati sotto forma di **Flow**, flussi di dati reattivi che trasportano informazioni in tempo reale. Questi Flow vengono poi passati agli Use Case, che li elaborano e li restituiscono all'**HomeViewModel**.

L'HomeViewModel converte i Flow ricevuti dagli Use Case in **StateFlow**. Gli StateFlow sono una tipologia di Flow "**hot**", ovvero sempre aggiornati e che emettono elementi indipendentemente dal fatto che qualcuno stia ascoltando. Questi garantiscono una perfetta sincronizzazione tra i dati e l'interfaccia utente.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

L'interfaccia utente effettua la collezione degli StateFlow, convertendoli in **Compose State**. In questo modo, l'interfaccia reagisce istantaneamente ai cambiamenti dei dati, aggiornando le informazioni visualizzate in tempo reale.

Quando l'utente preme sul pulsante per avviare il tracking della camminata, l'applicazione deve gestire i permessi necessari per accedere ai dati di localizzazione, gestire le notifiche, accedere al sensore di rilevamento dell'attività fisica.

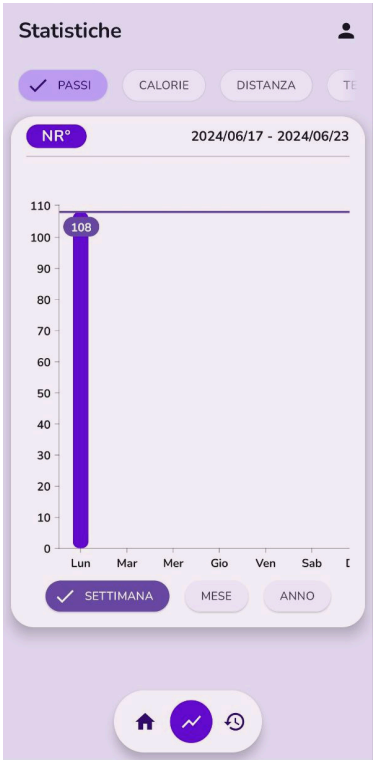
I permessi sono gestiti attraverso una coda che mantiene traccia dei permessi da richiedere e di quelli già rifiutati. La richiesta dei permessi avviene tramite un **LauncherForActivityResult**.

Quando un permesso viene rifiutato, viene mostrato un **dialog** che consente all'utente di richiedere nuovamente il permesso. Se il permesso viene rifiutato una seconda volta, non può più essere richiesto dall'applicazione; in questo caso, è necessario abilitarlo manualmente nelle impostazioni dell'app. A tal fine, viene mostrato un dialog che reindirizza l'utente direttamente alla pagina corretta delle impostazioni per abilitare il permesso.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.4 Statistics

Questa feature offre una panoramica completa dei progressi effettuati, presentando i dati monitorati in modo chiaro e interattivo. Utilizziamo la libreria open-source **Vico**, perfettamente compatibile con Jetpack Compose, per creare grafici a barre dinamici che visualizzano le varie metriche monitorate in fase di tracking.
Sono presenti alcune **chips** selezionabili che ti permettono di cambiare la metrica visualizzata (passi, distanza, calorie bruciate, velocità, tempo) e l'intervallo di tempo del grafico (settimana, mese o anno).



Lo **StatsViewModel** gestisce tutte le interazioni dell'utente nella schermata delle statistiche attraverso l'unica funzione pubblica **onEvent()**, che accetta come parametro un'evento di tipo **StatsEvent**.

All'inizializzazione, lo StatsViewModel richiama funzioni private per recuperare i dati relativi alla metrica e all'intervallo selezionati. Le informazioni vengono mantenute in un **MutableStateFlow** che contiene una classe **UiStatsState**. Questa classe racchiude i dati da visualizzare nella schermata delle statistiche. Questa è in grado di accedere alla versione read-only del MutableStateFlow, e quindi al corrispondente **StateFlow**, per poter reagire ai cambiamenti dei dati..

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Questa feature prevede di far mostrare le metriche monitorare facendo utilizzo di un chart a barre creato attraverso la libreria open-source **Vico**, la quale si integra perfettamente con Compose. La **StatsScreen** mostra anche delle chips selezionabili che permettono di cambiare la metrica visualizzata e l'intervallo di visualizzazione del grafico (settimana, mese o anno).

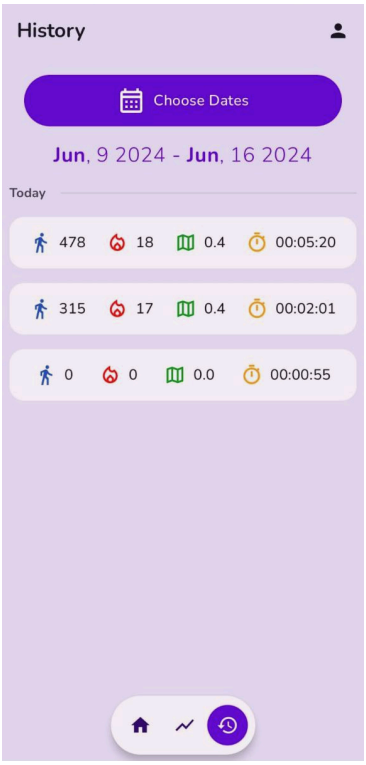
Un **LaunchedEffect** nella StatsScreen manipola i dati per renderli compatibili con la libreria dei grafici VIco.

Quando viene selezionata una chip, viene lanciato uno StatsEvent che attiva la funzione di recupero dati. Questo avviene attraverso specifici **Use Case**, in base alla metrica e all'intervallo selezionati. Gli Use Case interagiscono con il **WalkRepository**, che gestisce l'accesso ai dati delle camminate. Il recupero dei dati avviene in coroutine dedicate che utilizzano il dispatcher di I/O per ottimizzare le prestazioni.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.5 History

Questa feature permette all'utente di visualizzare la cronologia delle sue camminate, ordinate cronologicamente e corredate di statistiche complete. È possibile selezionare un intervallo di date per filtrare le camminate da visualizzare. Cliccando su una camminata, si apre un popup con le statistiche dettagliate e una mappa del percorso effettuato.



La schermata principale presenta un selettore di date per scegliere l'intervallo di tempo desiderato e un elenco sottostante che mostra le camminate all'interno del range selezionato.
Per il selettore è stato utilizzato il **DatePicker**, un componente di Material3 che permette di scegliere un range di date tramite un calendario.

Per ottenere l'elenco delle camminate, durante l'inizializzazione del **HistoryViewModel,** viene eseguita una coroutine nel dispatcher I/O che esegue uno **Use Case** per recuperare tutte le camminate nel range di date selezionato (inizialmente impostato su un valore predefinito) dal **DataBase**, insieme ai relativi **PathPoint** che rappresentano il percorso seguito dall'utente durante la camminata.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Lo Use Case preleva tutte le camminate dal DataBase e le raggruppa per data creando una lista di oggetti di tipo **AllWalksPerDate**. Questo è composto da una data ed una lista di oggetti di tipo **WalkWithPathPoint**. Verrà quindi restituita all'HistoryViewModel una lista di date con associate tutte le camminate relative a quella data.

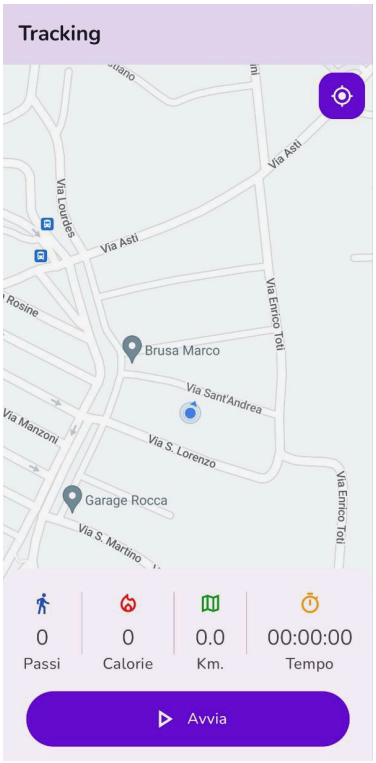
Tutte le camminate ottenute vengono utilizzate per effettuare un update del **MutableStateFlow** che mantiene la lista di camminate. In questo modo, potranno essere visualizzate correttamente nella **History Screen**.
Quando l'utente modifica il range di date, viene lanciato un evento di tipo **HistoryEvent** tramite la funzione **onEvent()** definita nel HistoryViewModel. Questa funzione gestisce gli eventi UI relativi a questa feature ed è l'unica funzione esposta all'UI e perciò è utilizzata per gestire tutte le interazioni dell'utente.

La funzione onEvent() avvia una coroutine, la quale chiama lo Use Case che si occupa di ottenere le camminate aggiornate dal DataBase utilizzando le nuove date selezionate come parametro. Una volta ottenute le camminate, viene effettuato un update del MutableStateFlow che mantiene l'elenco aggiornato delle camminate. Di conseguenza l'History Screen si aggiorna automaticamente per mostrare il nuovo elenco di camminate.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.6 Tracking

La feature di Tracking è quella che di fatto implementa i principali requisiti funzionali dell'applicazione. La schermata **TrackingScreen** mostra una mappa con il percorso attualmente compiuto, una card con le metriche tracciate (ad esempio passi, distanza, calorie bruciate ecc), e pulsanti per avviare, mettere in pausa e terminare il tracciamento.



All'avvio della TrackingScreen, il **TrackingViewModel** effettua una serie di operazioni utili per il corretto funzionamento di questa feature. Utilizza una serie di **Use Case** per recuperare i dati riguardanti l'altezza e il peso dell'utente salvati nel **Preferences DataStore**, dato che questi dati risultano essere fondamentali per il calcolo delle calorie.

Durante la fase di inizializzazione, viene verificato se il GPS è abilitato e il relativo **Compose State** esposto all'UI viene aggiornato di conseguenza. Se il GPS non è attivo, la TrackingScreen, tramite un **LaunchedEffect**, specifica una callback che lancia una **IntentSenderRequest** per notificare l'utente e richiedere l'abilitazione del GPS. Questa verifica avviene anche quando il monitoraggio della camminata viene avviato, registrando un **Broadcast Receiver** che risponde agli eventi di sistema relativi all'abilitazione o disabilitazione della localizzazione, aggiornando il Compose State che mantiene l'informazione sullo stato del GPS. Di conseguenza, verrà eseguita una recomposition che riattiverà il LaunchedEffect, il quale richiederà eventualmente all'utente di abilitare il GPS. Se l'utente rifiuta, il monitoraggio della camminata, se attivo, verrà messo in pausa. Tale Broadcast Receiver verrà rimosso nell'**onDispose()**.

Durante questa fase iniziale, viene anche richiesta l'ultima posizione rilevata per permettere alla mappa di zoomare automaticamente sulla zona in cui si trova l'utente.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Per adempiere a queste ultime due operazioni viene utilizzato, direttamente o indirettamente tramite uno **Use Case**, il **LocationTrackingManager**, un gestore che si occupa di tutte le operazioni relative ai dati di posizione.

Anche in questo caso, tutte queste operazioni di inizializzazione del TrackingViewModel vengono effettuate in diverse **coroutine** che eseguono in parallelo per ottimizzare il processo di recupero dei dati, sfruttando il dispatcher di I/O. Dopo ogni lettura, come già detto, vengono aggiornati i relativi **Compose State** esposti all'UI.

Il TrackingViewModel gestisce tutte le interazioni dell'utente nella TrackingScreen attraverso l'unica funzione pubblica **onEvent()**, che accetta come parametro un'evento di tipo **TrackingEvent**. Inoltre il TrackingViewModel espone all'UI un **MutableStateFlow** che contiene un oggetto di tipo **UiTrackingState**. Il suo compito è quello di mantenere tutti i dati di tracciamento, come le metriche e i vari **PathPoint** del percorso, in modo da poterli far visualizzare dalla TrackingScreen e permettere una reazione ai cambiamenti reattiva.

L'oggetto UiTrackingState viene aggiornato basandosi su un riferimento in sola lettura ad uno **StateFlow**, contenuto nella classe di tipo "state holder" chiamata **WalkHandler**, che contiene i dati effettivamente manipolati dal **Foreground Service** riguardanti lo stato della camminata. Il collezionamento di questo StateFlow per aggiornare lo UiTrackingState avviene in modo lifecycle-aware, assicurando che il flow non venga collezionato quando la TrackingScreen non è visibile in primo piano. In questo modo, si evita di eseguire calcoli inutili per l'aggiornamento della mappa nella TrackingScreen quando l'utente non la sta visualizzando.

Le operazioni di avvio, pausa, ripresa e terminazione del monitoraggio della camminata sono gestite attraverso specifici **Use Cases**, che verranno eseguiti quando la funzione onEvent() riceve il relativo evento. Questi Use Cases interagiscono con il **TrackingServiceManager**, un componente responsabile di inviare **intent**, in cui vi è associata una specifica azione, al Foreground Service. Il TrackingServiceManager facilita la comunicazione con il Foreground Service, assicurando che le operazioni vengano eseguite correttamente.

Le operazioni offerte dal TrackingViewModel per monitorare la camminata sono:

- **Avvio**: Invia un intent per avviare il Foreground Service, che inizia a monitorare la posizione, i dati dei sensori ed il tempo trascorso..
- **Pausa**: Invia un intent per mettere in pausa il monitoraggio, mantenendo lo stato attuale senza raccogliere nuovi dati.
- **Ripresa**: Invia un intent per riprendere il monitoraggio, continuando a raccogliere dati da dove si era interrotto.
- **Terminazione**: Invia un intent per terminare il Foreground Service, e quindi il monitoraggio, e successivamente salva i dati raccolti della camminata nel **Database**

Queste operazioni vengono avviate in coroutines, sempre per questioni di performance.

Come menzionato precedentemente, il **WalkHandler** è un componente chiave che mantiene lo stato del tracciamento della camminata. Mantiene tutte le informazioni sullo stato della camminata, come le metriche tracciate e i punti di percorso (PathPoint). Esso offre varie operazioni per l'aggiornamento dello stato, garantendo che siano thread-safe per prevenire race

condition e inconsistenze. Il Foreground Service aggiorna continuamente il **MutableStateFlow** contenuto all'interno del WalkHandler con i nuovi dati ottenuti, mentre il TrackingViewModel legge lo StateFlow esposto per aggiornare in tempo reale lo stato della UI.

Il Foreground Service è cruciale per il tracciamento in background, ed eseguirà alcune operazioni in base all'azione specificata nell'intent ricevuto, infatti a seconda dell'azione passata nell'intent (avvio, pausa, ripresa, terminazione), il servizio esegue l'operazione corrispondente. Viene utilizzato il **LocationTrackingManager** per raccogliere dati di posizionamento e il **TimeTrackingManager** per monitorare il tempo trascorso dall'inizio della camminata. Il LocationTrackingManager utilizza il **Fused Location Provider** per effettuare una richiesta di posizione ogni secondo, e permette di ottenere un Flow di **Location**.

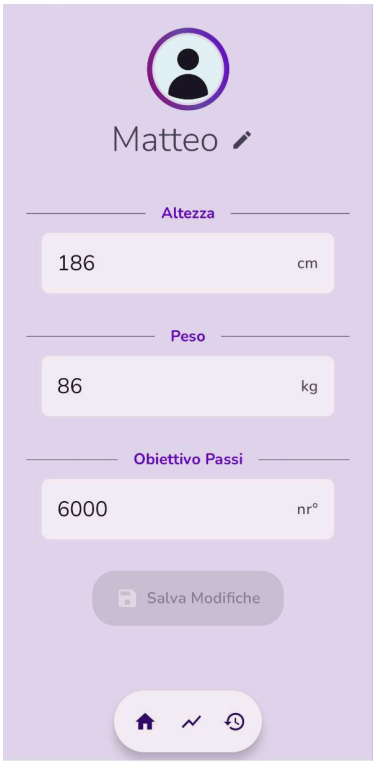
Il Foreground Service avvia anche il monitoraggio dei dati relativi ai sensori come lo **Step Counter** e l'**Accelerometro** (quest'ultimo utilizzato solo se lo Step Counter non è disponibile). I sensori vengono gestiti tramite un'interfaccia definita, chiamata **AndroidSensor**, che fornisce funzioni per installare callback e ottenere dati dei sensori in modo elegante. Tutte queste operazioni vengono eseguite all'interno di coroutine, migliorando l'efficienza e la reattività del servizio.

Per assicurare che Android non termini il processo, e quindi il tracciamento, anche quando l'applicazione non è in foreground, viene mantenuta attiva una **notifica** persistente. La notifica informa l'utente che il tracciamento è in corso e impedisce la chiusura del processo dell'applicazione da parte del sistema operativo. Mostra anche alcune delle metriche tracciate in tempo reale. Per gestire e aggiornare questa notifica, viene utilizzato il **NotificationManager**. Alla notifica è associato un **PendingIntent** che, al clic, riapre l'applicazione, permettendo all'utente di tornare facilmente alla TrackingScreen.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

5.7 Profile

La feature Profile permette di modificare e aggiornare i dati inseriti durante la fase iniziale di onboarding dell'app. La schermata **ProfileScreen** presenta una serie di campi di testo pre-compilati con le informazioni attuali. Queste informazioni possono essere modificate facilmente per mantenere il profilo aggiornato.



All'avvio della ProfileScreen, il **ProfileViewModel** utilizza una serie di **Use Case** per recuperare i dati salvati nel **Preferences DataStore**. Diverse **coroutine** lavorano in parallelo per ottimizzare il processo di recupero, sfruttando il dispatcher di I/O per operazioni di input e output efficienti. Dopo ogni lettura, viene aggiornato il **Compose State** esposto all'UI, che contiene un oggetto tipo **UiProfileState**. Il suo compito è quello di mantenere i dati e gli eventuali errori riguardanti i vari campi della ProfileScreen.

Quando l'utente modifica uno dei campi, viene lanciato uno specifico evento di tipo **ProfileEvent** tramite la funzione **onEvent()** definita nel ProfileViewModel. Questa funzione gestisce gli eventi UI relativi a questa feature ed è l'unica funzione esposta all'UI e perciò è utilizzata per gestire tutte le interazioni dell'utente.

In questo caso, tale funzione richiama le corrispondenti funzioni private del ProfileViewModel, che andranno ad aggiornare il valore del Compose State esposto. Viene anche eseguita una validazione del testo inserito richiamando gli specifici **Use Case**, per controllare la presenza di eventuali errori. La presenza di un errore comporta anch'esso l'aggiornamento del Compose

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

State, in modo tale da scatenare una recomposition per mostrare l'errore all'utente. Tutto ciò è eseguito in una coroutine, in modo da evitare il blocco dell'UI Thread.

Quando non ci sono errori e l'utente procede premendo il pulsante "Save Changes", viene lanciato un nuovo evento, il quale avvierà una serie di coroutines che si occuperanno di salvare le informazioni dell'utente nel **DataStore** e l'obiettivo di passi giornaliero nel **Database**, il tutto eseguendo gli opportuni **Use Cases**.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

6 TEST

Il processo di test dell'applicazione è stato articolato e rigoroso, coinvolgendo più fasi e diverse tipologie di test per garantire un'alta qualità del prodotto finale.

Test interni

- **Unit testing**: Sono stati scritti test unitari per verificare il corretto funzionamento di ogni singola parte del codice. I test unitari sono stati applicati a metodi e funzioni individuali per garantire che ciascun componente dell'applicazione funzionasse correttamente in isolamento. Ogni test ha verificato diversi scenari di input e output, coprendo casi sia tipici che limite. Questo ha permesso di isolare e correggere bug e garantire che ogni parte del codice soddisfi i requisiti.
- **Instrumented testing**: Successivamente, sono stati effettuati test strumentali per verificare il corretto funzionamento delle operazioni che utilizzano l'API del Framework Android. In particolare, sono stati eseguiti test approfonditi su tutte le funzionalità legate al database, assicurandosi che le diverse funzioni del DAO restituissero correttamente i valori richiesti. Sono stati scritti test sia per il DAO relativo alle camminate, sia per quello relativo agli obiettivi di passi giornalieri. Inoltre, sono stati testati accuratamente i meccanismi del DataStore per garantire che i valori restituiti fossero corretti e che le funzioni di aggiornamento dei dati funzionassero come previsto.
- **System testing**: Una volta verificato il corretto funzionamento dei singoli componenti, sono stati condotti test di sistema per valutare l'intera applicazione come un unico sistema. Sono stati eseguiti test molto approfonditi dell'applicazione, che hanno permesso di individuare e risolvere eventuali bug e apportare le necessarie correzioni prima di passare alla fase successiva, ovvero il beta testing.

Test esterni

- **Beta testing**: Dopo aver completato i test interni, abbiamo richiesto ad alcuni amici e familiari di testare l'applicazione. Ogni "beta tester" ha utilizzato l'applicazione per un periodo di tempo, monitorando le proprie camminate e fornendo feedback dettagliati su vari aspetti dell'applicazione.
- **Feedback e miglioramenti**: Dai test condotti dai beta tester, sono emersi diversi punti critici e suggerimenti per migliorare l'applicazione. Tra i feedback più frequenti, è stata evidenziata la necessità di migliorare la precisione del tracciamento, che in alcuni casi causava registrazioni inaccurate dei percorsi finali. Altri punti rilevati includono l'usabilità dell'interfaccia utente, la chiarezza delle statistiche visualizzate e la risoluzione di alcuni bug grafici, particolarmente evidenti su dispositivi con differenti caratteristiche di display. In risposta a questi feedback, abbiamo effettuato modifiche significative all'applicazione, risolvendo i bug segnalati e ottimizzando l'esperienza complessiva dell'utente.

Test di usabilità

- **Sessioni di test di usabilità**: Abbiamo condotto ulteriori test di usabilità con amici e familiari. Questi utenti sono stati osservati mentre utilizzavano l'applicazione senza alcun tutorial, per completare vari compiti, come iniziare e terminare una camminata, visualizzare il percorso su mappa, e consultare le statistiche settimanali. Durante queste sessioni, abbiamo raccolto osservazioni sui comportamenti degli utenti e identificato eventuali problemi riscontrati nell'utilizzo dell'app.
- **Analisi dei risultati**: Le sessioni di test di usabilità hanno fornito preziose informazioni su come migliorare ulteriormente l'interfaccia utente e l'esperienza complessiva. In particolare, abbiamo apportato modifiche per rendere più intuitivi alcuni elementi dell'interfaccia e per semplificare l'accesso alle funzionalità principali.

In conclusione, il processo di test dell'applicazione è stato complesso e multifase, coinvolgendo sia test interni rigorosi che feedback da parte di utenti esterni. Questo approccio ha garantito che l'applicazione fosse robusta, facile da usare e in grado di soddisfare le esigenze degli utenti finali.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

7 CONCLUSIONI

Il progetto ha soddisfatto pienamente i requisiti iniziali, come dimostrato dall'analisi delle funzionalità implementate e dai risultati dei test eseguiti. Di seguito, analizziamo come ciascun requisito sia stato rispettato e come i test abbiano confermato la correttezza e l'efficacia della soluzione finale.

7.1 Rispetto dei requisiti iniziali

L'applicazione sviluppata rispetta tutti i requisiti iniziali che sono stati prefissati.

Tracciamento delle camminate

- **Requisito**: L'applicazione deve tracciare le camminate degli utenti, monitorando passi, calorie, chilometri e tempo totale.
- **Implementazione**: È stata integrata una funzionalità di tracciamento tramite i Fused Location Provider e sensori di movimento per rilevare i passi. Le calorie bruciate sono calcolate in base ai dati raccolti. La distanza percorsa e il tempo totale dell'attività vengono registrati e visualizzati in tempo reale.
- **Verifica**: I test hanno dimostrato che il tracciamento è preciso e coerente, confrontando i risultati con altre app di tracciamento.

Visualizzazione del percorso su mappa

- **Requisito**: Il percorso effettuato deve essere visibile su mappa.
- **Implementazione**: Utilizzando le API di Google Maps, l'app mostra il percorso su una mappa interattiva, aggiornandolo in tempo reale.
- **Verifica**: I test hanno confermato che i percorsi sono correttamente tracciati e visualizzati senza interruzioni o errori.

Statistiche settimanali, mensili ed annuali

- **Requisito**: Gli utenti devono poter consultare grafici con le statistiche delle attività.
- **Implementazione**: Sono stati creati grafici interattivi che mostrano le statistiche settimanali, mensili ed annuali per passi, calorie, chilometri e tempo di attività.
- **Verifica**: I test sui grafici hanno dimostrato che i dati vengono aggregati e visualizzati correttamente per i diversi intervalli di tempo.

Cronologia delle camminate

- **Requisito**: L'applicazione deve avere una cronologia delle camminate con tutte le relative statistiche e i percorsi su mappa organizzate in ordine cronologico in base alla data. L'utente deve avere la possibilità di aggiornare il range di date in cui mostrare le camminate.
- **Implementazione**: È stata sviluppata una sezione di cronologia dove gli utenti possono visualizzare tutte le camminate passate con i dettagli statistici e i percorsi mappati, possono utilizzare un DatePicker per selezionare un particolare range di date..

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- **Verifica**: I test sulla cronologia hanno confermato che tutte le attività vengono salvate e possono essere consultate in modo accurato e veloce e le date mostrate sono corrette rispetto a ciò che ha selezionato l'utente.

Interfaccia utente

- **Requisito**: L'interfaccia utente deve essere intuitiva e facile da usare.
- **Implementazione**: È stata sviluppata un'interfaccia utente moderna e user-friendly, utilizzando il framework Jetpack Compose per garantire una UI reattiva e fluida.
- **Verifica**: Le sessioni di test di usabilità con amici e familiari hanno mostrato che gli utenti sono in grado di navigare e utilizzare l'app senza difficoltà.

Persistenza dei dati

- **Requisito**: I dati dell'utente devono essere salvati in modo persistente per essere recuperati successivamente.
- **Implementazione**: È stato utilizzato il Preferences DataStore per salvare i dati in modo persistente e sicuro.
- **Verifica**: I test hanno dimostrato che i dati vengono correttamente salvati e recuperati, anche dopo il riavvio dell'applicazione.

Manutenibilità

- **Requisito**: Il codice deve essere facilmente mantenibile e aggiornabile.
- **Implementazione**: Il progetto è stato strutturato seguendo i principi della Clean Architecture, separando le responsabilità in moduli distinti e utilizzando pattern di progettazione MVVM (Model-View-ViewModel).
- **Verifica**: La struttura modulare del codice ha facilitato l'aggiunta di nuove funzionalità e la risoluzione di bug, migliorando la manutenibilità del progetto.

Reattività e performance

- **Requisito**: L'applicazione deve essere reattiva e performante, rispondendo rapidamente alle interazioni dell'utente.
- **Implementazione**: Sono stati utilizzati Flow, StateFlow, Compose State e Coroutines per gestire lo stato dell'app e le operazioni asincrone in modo efficiente. Queste tecnologie consentono di eseguire operazioni in background senza bloccare il thread principale, migliorando la reattività dell'applicazione.
- **Verifica**: I test di utilizzo hanno confermato che l'app è reattiva e fluida, con tempi di risposta rapidi.

7.2 Conferma tramite test

Per garantire che il progetto rispondesse ai requisiti, sono stati eseguiti vari test, tra cui:

Unit Tests

- **Obiettivo**: Verificare il corretto funzionamento delle singole unità di codice (metodi, classi).
- **Risultati**: Tutti i test unitari sono stati superati, confermando la correttezza delle funzioni.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Instrumented Tests

- **Obiettivo**: Testare le parti di codice che usufruiscono del Framework di Android
- **Risultati**: Tutti i test sono stati superati dimostrando la correttezza delle funzioni.

Beta test

- **Obiettivo:** Verificare il corretto funzionamento dell'applicazione nel complesso tramite il feedback di diversi utenti in modo da individuare eventuali bug e possibili miglioramenti.
- **Risultati:** I test hanno rilevato alcune criticità riguardanti l'interfaccia grafica e la precisione del tracciamento. Tutti i problemi riscontrati sono stati risolti nelle successive versioni dell'applicazione.

In conclusione, il progetto non solo ha rispettato tutti i requisiti iniziali, ma ha anche dimostrato attraverso test rigorosi di essere una soluzione affidabile e robusta per il tracciamento delle camminate e l'analisi delle attività fisiche. La combinazione di un design attento e un'implementazione accurata ha permesso di realizzare un'applicazione che soddisfa pienamente le aspettative degli utenti.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603